Published at the Society for Computer Simulation's Advanced Technology Simulation Conference, Seattle, WA April, 2001

Using Linear Genetic Programming to Develop a C/C++ Simulation Model of a Waste Incinerator

Larry M. Deschaine, Janardan J. Patel, Ronald D. Guthrie, and Joseph T. Grumski
(Science Applications International Corporation)
and M.J. Ades
For Information Contact:
Larry.M.Deschaine@alum.mit.edu

Abstract

We explore whether Linear Genetic Programming (LGP) can evolve a C/C++ computer simulation model that accurately models the performance of a waste incinerator. Human expert written simulation models are used worldwide in a variety of industrial and business applications. They are expensive to develop, may or may not be valid for the specific process that is being modeled, and may be erroneous.

LGP is a machine learning technique that uses information about a process's inputs and outputs to simultaneously write the simulation model, calibrate and optimize the model's constants, and validate the solution. The result is a calibrated, validated, error-free C/C++ computer model specific to the desired process.

To evaluate whether this is feasible for complex industrial processes, the method on data obtained from the operation of a hazardous waste incinerator. This process is difficult to model. Previously, in a well-conducted study, the popular machine learning technique, analytic neural networks, was unable to derive useful solutions to this problem. The present study uses various mutation rates (95%, 50%, and 10%), 10 random initial seeds per mutation rate, and a large number of generations (1,280 to 4,461). The LGP system provided accurate solutions to this problem with a validation data measure of fitness, R^2, equal to 0.961.

This work demonstrates the value of LGP for process simulation. The study confirms previously published results and found that the distribution of outputs from multiple genetic programming (GP) runs tends to include an extended "tail" of outstanding solutions. Such a tail was not found in previous studies of neural networks. This result emphasizes the need for employing a strategy of multiple runs using various initial seeds and mutation rates to find good solutions to complex problems using LGP. This result also demonstrates the value of a fast LGP algorithm implemented at the machine code level for both static scientific data mining and real-time process control. The work consumed 600 hours of CPU time; it is estimated that other GP algorithms would have required between 4 and 136 years of CPU time to achieve similar results.

INTRODUCTION

With the increasing complexity of modern manufacturing [Popovic98] and processes [Popovic90], industries require fast techniques for adaptive real-time control [Francone00a]. Today, many industries allocate in excess of 10% of their plant investment capital outlays for instrumentation and control [Murrill00]. This percentage has doubled over the past 30 years and shows no signs of diminishing. The industrial processes often are non-linear and the mathematical representation of the process is unknown [Sinha00]. Hence, simulation models are not available for many of the processes that exist today. Without simulation models, optimal process control is very difficult to achieve. This results in unnecessary waste of resources.

GP is a promising machine learning technology that has been the subject of intensive academic research since 1988. Since 1998, commercial applications have been made available on the market [Francone00b]. GP can be used to automatically develop a simulation model of complex industrial processes. In this work, we specifically examine whether a very fast form of GP, LGP can evolve a C/C++ computer simulation model that simulates the behavior of the concentrations of carbon dioxide in a waste incinerator. Waste incineration was chosen as a test case because it is a very complex process that involves variable input material properties (i.e., solids, liquids, and gaseous), high temperature, large temperature variations, variable-input energy sources, compressible gas flow, density effects etc. The simulated variable, the concentration of carbon dioxide in the secondary combustion chamber, varies widely from 0 to 5,000 parts per million (ppm). This process has been demonstrated resistant to solution by machine learning via the analytical neural network technique (ANN), as evidenced by a well-conducted study [Fausett00].

Despite the complexity of the incineration processes, developing a computer simulation code that maps the process variables to the carbon dioxide emission concentration should at least be feasible using GP. The strength of GP is its ability to abstract an underlying principle from a finite set of fitness cases [Banzhaf98, Koza99]. This principle can be considered the essence of the regularities that determine the appearance of concrete fitness cases. GP evolves both the structure and the constants of the solution simultaneously, the goal being to extract these

regularities and put them under the form of an algorithm or computer program, which then represents a simulation model.

THE GENETIC PROGRAMMING ALGORITHM

The Goal

The task given to the GP algorithm is to develop a C/C++ computer simulation program that accurately maps the important waste incineration process variables with the concentration of carbon dioxide in the waste incineration process. The goal of GP is to evolve the one excellent solution that solves the problem, as opposed to evolving many average or good solutions. Identifying the important inputs will reduce the cost of the real-time process monitoring and control system. A computer program that predicts the concentration of air contaminants can reduce the cost of monitoring the incineration facility and provide a tool to simulate emissions prior to incinerator loading, thereby improving compliance with the U.S. Environmental Protection Agency (EPA) Clean Air Act (CAA), as amended. An optimal strategy for incinerating waste, as defined by maximizing incineration rate and optimizing profits without operating permit violations, then can be developed.

The Genetic Programming Algorithm

For this work, a GP algorithm was used that specifically evolves a computer program at the machine code level [Nordin97&99] on a Von Neumann machine, a machine where the data and program resides in the same storage location. This approach is called LGP, and also is known as the Automated Induction of Machine Code by Genetic Programming (AIMGP). This system formally was known as the Compiling Genetic Programming System (CGPS). The algorithm, since it manipulates the machine code directly, avoids compilation overhead and has been estimated to be more than 60 times faster when compared to an interpreting C-language implementation and up to 1,500 to 2,000 times faster when compared to a LISP-computer language implementation. While any of the number of available GP algorithms that have been developed could have been used [Banzhaf98], the speed of LGP means that programs that would take months or years to evolve on a single processor can be evolved in less than a day. For example, this research consumed 600 hours of CPU time. A GP system that relied on C-language interpreter approach would have consumed approximately 4 years of CPU time and a LISP computer programming GP approach would have taken approximately 100 years.

Machine Learning Facilitating Human Learning

GP is a machine learning technique that writes computer programs. During a GP run, an intron explosion occurs. An intron is an instruction that occurs in a program that has little or no consequence on the output. Introns are believed to occur to protect a good program from the destructive effects of crossover [Banzhaf98]. To the human observer, these introns make the code confusing. Specific algorithms have been developed that transform the raw machine learned program to a human understandable program. These algorithms, which clean, simplify, and optimize the code, facilitate the process of transferring the knowledge derived during machine learning to the human expert, facilitating acceptance of the solution as well as increasing the knowledge of the process.

Parsimony Pressure

Parsimony pressure is a term used to refer to techniques that tend to make the evolved program shorter. Parsimony pressure causes “natural selection in evolutionary learning systems to favor the selection of shorter and more compact programs. It is a penalty function on program length. The function calculates the percentage difference between two programs evolved using the tournament selection algorithm. If the shorter program meets acceptance criteria when compared to competing individuals, the shorter program is selected as the winner. Shorter programs often are easier to interpret. Parsimony was not deemed necessary in these simulations, as discussed in the results section.

Intron Removal

Substituting the no operation instruction into the program and comparing the two output values remove introns. If the output value does not change, the instruction is deemed to be an intron and is removed.

Custom Fitness Functions

A custom fitness function can be incorporated that allows the researcher to specify precisely how the results of the GP evolutions are scored, in addition to the standard linear and least squares and R^2 error approach.

Interactive Evaluation of Evolved Program

Interactive evaluation of the evolved program allows the researcher to simplify, modify, and optimize the program as well as explore the results of the modification(s) on the program’s fitness. This algorithm also allows the researcher to perform what-if scenarios on the code, including importing solutions from known sources or other GP runs

THE TEST CASE

LGP generated excellent solutions to the incinerator problem. The incineration prediction challenge, however, had no known solution; hence a benchmark with which to compare either the results or the resulting program does not exist. The results of the program fit were compared to the validation data set to verify the accuracy of the results.

To test the LGP technique, the performance of a waste incineration plant was modeled using real-time typical operational data collected hourly over a 1-week period at an incinerator [Fausett00]. The incinerator processes a variety of solid and aqueous waste, using a combination of a rotary kiln, a secondary combustion chamber, and an off gas scrubber system. The process chosen is a very complex and consists of variable fuel and waste inputs, high temperatures of combustion, and high velocity offgas emissions. Variables that describe the incinerator process were collected as follows:

- Process Parameters
  - Rotary Kiln Incinerator (lb./hr): Fuel Oil (flow), Liquid Waste (flow), Solids (airflow), Solids (flow), Solids (average flow), Fuel Oil (airflow), Liquid Waste (airflow), Aqueous Waste (flow), Temperature (C), Time (hrs.)
  - Secondary Combustion Chamber (lb./hr): Fuel Oil (flow), Fuel Oil (airflow), Steam (flow), Temperature (C), percent O_2 .
  - Offgas: Four measurements of CO_2 (ppm), two measurements of percent O_2, and one measurement of duct flow (scfm).
- Output Parameter
  - Secondary Combustion Chamber: CO_2 (concentration as ppm).

If the GP is to be successful, it must develop the relationship that maps the variables of the incineration process to the carbon dioxide concentration in the secondary combustion chamber. This parameter was chosen because the ANN had great difficulty developing any useful model for this mapping [Fausett00]. We used a zero and 1 hour offset for the data when constructing the training and validation instance sets. This resulted in a total of 44 input variables. Thirty LGP simulations for a period of 20 hours each using 10 different random seeds for each of three mutation rates were run. The results of the LGP simulations were compared for solution convergence.

The LGP System Parameters

The parameters and values that were used in this test case, as well as the rationale for their selection, are discussed below.

Linear Genetic Programming

Population Size— A population in LGP is the number of programs that the system will evolve. Generally, the larger the population the more difficult the problem to be solved, and the longer the run will take. The size of the population is limited by the random access memory (RAM) of the computer. A population size of 25,000 was used, and the LGP algorithm consumed 95,692 Kbytes of RAM.

Maximum Number of Tournaments—A GP tournament is the process in which evolved programs are produced. The tournament algorithm is constructed as follows:

1. Initialize a Population of Programs. Create a population of randomly generated programs.
2. Perform the Tournament Contest. Randomly select four programs from the population and evaluate how well they map the input data to the output data. This step is known as the program "fitness" evaluation. Two programs are selected as winners and the other two are tagged as losers.
3. Transform the "Winner" Programs. The two "winner" programs then are copied and transformed probabilistically by:
   - Exchanging parts of the "winner" programs with each other to create two new programs (crossover), and/or
   - Randomly changing each of the tournament winners to create two new programs (mutation).
4. Replace the "Loser" Programs. Replace the "loser" programs in the population with the transformed "winner" programs. The winners of the tournament remain in the population unchanged.
5. Iterate Until Convergence. Repeat steps (2) through (4) until a program is developed that predicts the behavior sufficiently.

Time and memory only limit the number of tournaments. An arbitrarily large maximum number of tournaments of 2,147,483,647 was used, as the stopping criterion was 20 hours, this value was never reached.

Search Parameters:

- **Mutation Rate.** Mutation is one of the principal search operators used to transform programs in the GP algorithm. Mutation has the effect of causing random changes to occur in tournament winners. The mutation is applied probabilistically to all programs that have won tournaments, regardless of whether a winner has been selected for crossover. While many GP systems use either minor or no mutation operators, increasing the mutation rate has been shown to significantly improve the generalization capabilities of GP [Banzhaf96]. The mutation rate can range from 0% (no mutation to 100%). Mutation rate values of 95%, 50%, and 10% were selected. These values agree with reported research values, although the maximum mutation

rate used (95%) exceeds the 80% maximum reported value.

- **Crossover Rate.** The crossover rate is another key search parameter in LGP. The crossover operates by exchanging sequences of instructions between two tournament winners. This results in two programs being inserted into the population in place of the two losers in that tournament. The crossover rate can vary from 0% to 100%. We used a constant crossover rate of 50% for all 30 simulations.
- **Reproduction Rate.** Reproduction rate is another search operator in LGP. Reproduction copies a program and places the copy in the population in addition to the original program. It is a function of the crossover and mutation rates, as follows: reproduction_rate = 100-(mutation_rate)-(crossover_rate*[1-mutation_rate]).

Demes

- **Number of Demes.** The number of demes pertains to the division of the population of programs. A deme is a subset of the program population to essentially isolate groups of populations from each other. This paradigm is consistent with biologists' belief that genetic diversity is enhanced when populations are separated from each other geographically. A value of 10 demes for this run was selected, with 2,500 programs per deme.
- **Crossover Percentage Between Demes.** As discussed above, crossover occurs between programs in the same population. By dividing the population into demes, crossover now occurs both within each deme as well as between demes. The percentage of crossover between demes sets the percent of tournaments that will result in crossover between programs in adjacent demes. The algorithm works as follows:
  1. Select a deme at random.
  2. Select one of the two adjacent demes at random.
  3. Select two programs from each of the selected demes, the better of which is selected for crossover.
  4. Crossover the selected program from each deme. The offspring of the crossover replaces the two tournament losers.

We used a deme crossover rate of 10%; it can range from 0% to 100%. Note that this "between deme" crossover rate of 10% provides a different purpose than the crossover rate that occurs in the same population, which is the value of 50% discussed above.

- **Dynamic Subset Selection**. Dynamic subset selection uses a subset of the data that the LGP uses to develop the equation, the training set, to help evolve solutions that are more generalized. It works by periodically changing which subset of the training set is used for training purposes, and hence helps avoid memorization.
- **Target Subset Size.** This size controls the size of subset of the training set that will be used. The total data set consisted of 166 instances, 151 training instances and 15 validation instances chosen as every 10^{th} point, which gave a representative range of output values to train and validate the solution against. A value of 120 training instances out of the total training set of 151 was used for the target subset size. The value for the target subset size is very problem specific because this was a small data set, a relatively large percentage (80%) of it was chosen, on larger data sets with thousands of instances, it would not be unusual to use only a fraction of the data at any one time. The key factor is that the subset selected should be representative of the data set as a whole.
- **Selection by Age.** The algorithm keeps track of the usage of each individual training instance. The training set is chosen in proportion with the time since the training instance was last used. The least recent training instances are preferentially chosen during the next training set assembly. A weight selection of 50% was chosen for this parameter.
- **Selection by Difficulty.** The algorithm also keeps track of how difficult it is for the LGP to find a good solution for a particular training instance of the training data set. By setting this parameter as non-zero, more general solutions are found. A selection weight of 50% was chosen for this parameter.
- **Stochastic Selection.** This parameter is used to select training instances randomly. A selection weight for this criterion was set to 0% because stochastic selection was not used in our runs.
- **Training Subset Change Frequency Equivalents.** This parameter determines how frequently the training subset is changed in "generation equivalents." Since the LGP uses the tournament selection criteria, a generation equivalent is one-half of the population size, in the case considered 12,500 tournaments. A value of (1) generation equivalent, or 12,500 tournaments, was used on the training data subset in these runs. Every 12,500 tournaments, a new training subset was developed according to the methodology described above.

RESULTS

The GP algorithm produced a very good solution to the incineration simulation challenge. Table 1 summarizes the results. The table summarizes the information from the best-validated program from each simulation.

Table 1. Comparison of R2 for Various Mutation Rates and Random Seeds

Fitness Case (R^2)	Mutation Rate = 95%	Mutation Rate = 50%	Mutation Rate = 10%
Validation Data Set			
Best Validated Fitness	0.961	0.785	0.852
Avg. Validated Fitness	0.720	0.477	0.552
Worst Validated Fitness	0.477	0.088	0.168
Complete Data Set (Validation and Training)			
Best Fitness	0.979	0.903	0.912
Avg. Fitness	0.709	0.569	0.531
Worst Fitness	0.483	-0.252	-0.117

The stopping criterion for all simulations was 20 hours. The average number of generations that were evolved was 3,037 (37,962,500 tournaments). The number of generations ranged from 1,280 (16,000,000 tournaments) to 4,461 (55,762,500 tournaments). Each tournament represents a new program being genetically designed for consideration as the best (i.e., most fit) computer program. It is clear to see that the shear speed of the LGP technique uses brute force to find the high fitness solutions.

The fitness of the solution was calculated by comparing the measured and predicted values using the linear trend-line option that is in Microsoft Excel®, with the y-intercept set to zero. The best-validated data results (R2 of 0.961) were obtained using a high mutation rate of 95%. Moreover the two best solutions were obtained at the 95% mutation rate. As Table 1 shows, the mutation rates of 50% and 10% failed to produce an R^2 validated fitness above 0.852, and had far worse average solution fitnesses. This is thought to have occurred because a high mutation causes a more aggressive search of the solution space as opposed to lower mutation rates. The best solution is presented in Figure 1. The distribution of validated solution fitness is shown in Figure 2. This solution then becomes the simulation tool from which to predict the waste incinerator performance and optimize it's operation, as shown in Figure 3.

The LGP algorithm was run 30 times for a total of 600hrs 09min 30sec of CPU time. The computer consisted of a dual Pentium III © 533Mhz PC with a 133 FSB, 256MB of ECC 100 RAM, the Intel © 840 chipset on a Supermicro PIIIDME motherboard using the Microsoft Windows 2000 Professional operating system. The LGP algorithm consumed approximately 95,692 KB of memory. The total memory requirement, including the operating system, was 204,784 KB when run as a single application.

CONCLUSIONS AND SUGGESTIONS FOR FURTHER RESEARCH

Earlier research suggests that, for three machine-learning problems, multiple runs on linear GP systems tend to produce a distribution of outputs with a long tail of outstanding quality solutions [Francone96]. Interestingly, this long tail of outstanding solutions exists even though the average solution generated may not be very good. By way of contrast, that same study found that neural networks produce a large number of runs that are normally distributed around a mean of average quality solutions, but that multiple neural network runs do not produce the tail of outstanding solutions characteristic of LGP.

The present study is consistent with the earlier results in [Francone96]. Like the previous study, neural network technology was unable to generate outstanding solutions to the problem at hand. And, like the previous study, multiple LGP runs produced a tail of outstanding solutions, which included solutions considerably better than any solution produced by neural network technology. To wit, LGP evolved a validated program that explains 96.1% [Validated R^2 of 0.961 from Table 1] of the relationship between the process inputs and output variable for a machine learning unfriendly incineration process.

In real world applications, the goal of machine learning is normally to produce the one super-solution that solves the problem as opposed to generating many average solutions. Two studies (including this present study) now suggest that an LGP approach meets this criterion better than do neural networks. GP performed extremely well on this very difficult process control problem. However, this technique did not perform well for all of the runs. This confirms the difficulty of the problem.

The results of this research also demonstrate the importance of solving problems by performing multiple LGP runs at different mutation rates and initial randomly

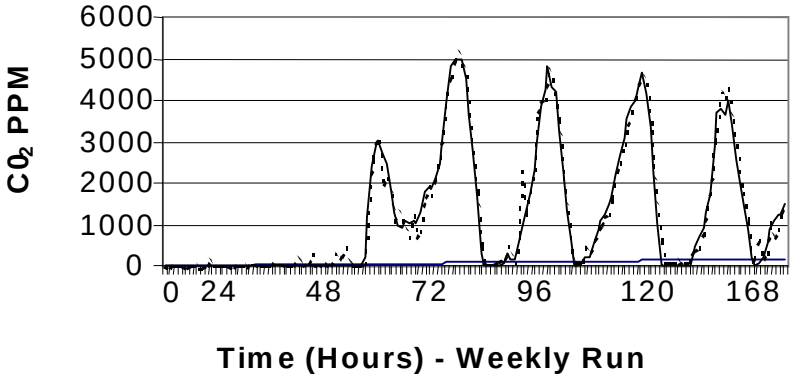

Figure 1. Predicted Versus Actual Emissions, $R^2 = 0.961$

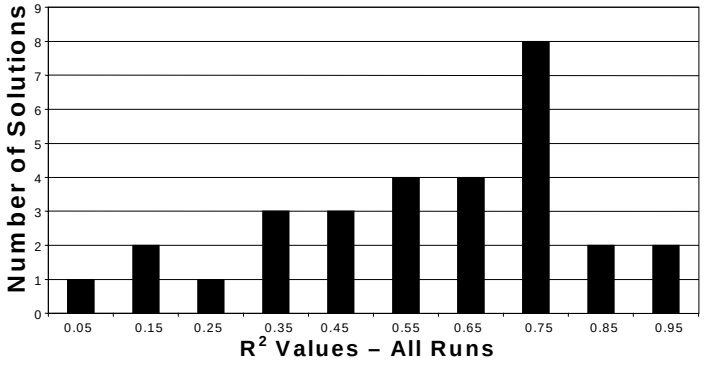

Figure 2. Distribution of GP Validation R^2

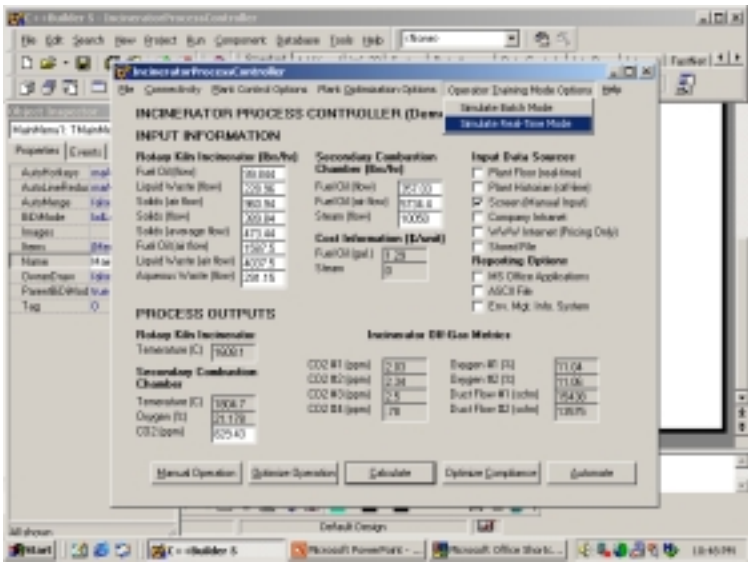

Figure 3. The Machine Learned Solution Deployed in a Windows®-based Simulation Tool

generated populations. This process facilitates finding the best solution because of the unique distribution of the results of multiple, LGP runs discussed above. However, multiple runs are very time consuming because machine learning in general (and GP in particular) is very computationally intensive. Ordinary GP or neural network software would not have been capable of performing the runs in this study in remotely realistic time frames, the next best GP algorithm would have consumed approximately 4 years of CPU time.

Accordingly, these conclusions underline the importance of LGP's great speed in solving process control problems. Process control frequently involves difficult learning domains that benefit from multiple runs. Thus, LGP's speed may well open other, historically intractable dynamic control of complex process problems to widespread solution by machine learning techniques.

This success demonstrates the promise of using LGP algorithms to simulate the waste incineration process. Further work includes testing this technique on other complex manufacturing processes. To date, success has been demonstrated on manufacturing of solid materials [Deschaine00a] and other environmental [Deschaine00b] processes.

With the simulation model developed, what-if scenario simulations of a planned waste incineration activity can help keep emissions to within acceptable limits. The C/C++ process description also allows process optimization to be explored via linkage with cost information and other management resource planning and optimization tools. Reducing the number of monitoring locations for data collection by focusing on the evolved solution inputs has obvious benefits with respect to capital expenditures, reduced operations and maintenance of the control system, and automated reporting.

ACKNOWLEDGEMENTS

The authors greatly acknowledge and thank Dr. Wolfgang Banzhaf, Dr. Peter Nordin and SAIC colleagues and managers: Dr. Clinton W. Kelly III, Mr. Joseph W. Craver III, Mr. John Aucella, Ms. Marita Fernald, Mr. Richard Pierce, and Dr. Jenifer McCormack. We also thank Dr. Laurene Fausett of USC-Aiken and Mr. Frank D. Francone and Mr. Edward Dilkes of Register Machine Learning Technologies. They all are thanked for the latitude, insightful comments, and freedom to explore this new and valuable technology. Register Machine Learning Technologies also is thanked for providing the LGP system used in this work, which reduced the CPU time of this research project from several years to approximately 1 month of CPU time, without such speed this research would not have been possible.

REFERENCES

Banzhaf98. Banzhaf, Nordin, Keller, Francone, Genetic Programming – An Introduction. On the Automatic Evolution of Computer Programs and its Applications. 1998 Morgan Kaufmann Publishers, Inc. 470 pp.

Banzhaf96. Banzhaf, W., Francone, F., and Nordin, P. (1996). *The Effect of Extensive Use of the Mutation Operator on Generalization in Genetic Programming Using Sparse Data Sets.* In Voigt, H., Ebling, W., Rechenberg, I., and Schwefel, H., P., editors, Parallel Problem Solving From Nature IV. Proceedings of the International Conference on Evolutionary Computation, Vol. 1141 of Lecture Notes in Computer Science, pages 300-310, Berlin. Springer-Verlag, Berlin.

Deschaine00a. Deschaine, L. M., Zafran, F. A., Patel, J. J., Amick, D., Pettit, R., SAIC, Francone, F. D., Nordin, P., Dilkes, E., and Fausett, L. V., *Solving the Unsolved-Using Machine Learning to Model a Complex Production Process – Case Example Applying Three Machine Learning Techniques*, Society for Computer Simulation's Advanced Simulation Technology Conference, Washington, DC, USA April 2000.

Deschaine00b. Deschaine, L. M., *Tackling Real-World Environmental Engineering Challenges with Linear Genetic Programming*, PCAI Magazine, September/October, 2000, pages 35-37.

Fausett00. Fausett, L. V., *A Neural Network Approach to Modeling a Waste Incinerator Facility*, Society for Computer Simulation's Advanced Simulation Technology Conference, Washington, DC, USA April 2000.

Francone96. Francone, Nordin, J.P. and Banzhaf W. (1996) Benchmarking The Generalization Capabilities of a Compiling Genetic Programming System Using Sparse Data Sets. In: *Proceedings of The First International Conference on Genetic Programming*, Stanford, USA

Francone00a. Francone, F. D., Nordin, P., Banzhaf, W. Dilkes, E., Deschaine, L. M. *AIM Learning™ Adaptive, Real-Time, Control Technologies.* Society for Computer Simulation's Advanced Simulation Technology Conference, Washington, DC, USA April 2000.

Francone00b. Francone, F. D. et al., *Discipulus Owners Manual*, Register Machine Learning Technologies, Inc. Version 2.0 2000.

Koza99. Koza, Bennett, Andre, Keane, *GENETIC PROGRAMMING III – Darwinian Invention and Problem Solving*, 1999 Morgan Kaufmann Publishers, Inc. 1154 pp.

Murrill00. Murrill, Paul W., *Fundamentals of Process Control Theory, 3rd Edition*, ISA: the International Society for Measurement and Control, North Carolina, USA, 333 pp.

Nordin97. Nordin, Peter, *Evolutionary Program Induction of Binary Machine Code and its Applications*, Krehl Verlag, Munster, 1997, 290 pp.

Nordin99. Nordin, P., Banzhaf, W., Francone, F.D., *Efficient Evolution of Machine Code for CISC Architectures Using Instruction Blocks and Homologous Crossover, Advances in Genetic Programming Vol. III*, The MIT Press, 1999 Chap. 12.

Popovic98. Popovic, D. and L. Vlacic, *Mechatronics in Engineering Design and Product Development*, Marcel Dekker, New York, 1998.

Popovic90. Popovic, D. and V.P. Bhatkar, *Distributed Computer Control for Industrial Control*, Marcel Dekker, New York, 1990.

Sinha00. Sinha, N.K., and Gupta, M.M., *Soft Computing & Intelligent Systems, Theory and Applications*, Academic Press, UK, Chapter 18 (D. Popovic), 2000.